MICROPROCESSOR HARDWARE AND SOFTWARE

G. Musstopf

SCS Scientific Control Systems, D-2000 Hamburg, F.R.G.

ABSTRACT. The present semiconductor components are significantly influencing the hardware- as well as the software-architecture of computer-based process control systems. Additionally they are opening new application areas for automation. On the other hand the supplement m i c r o is used too often and with very different meanings. Also the discussion between the three parties: hardware specialists, software specialists and users is getting more complicated instead of easier. For example a tremendous number of new technical terms is used without a generally agreed upon definition.

Looking to process control applications - especially for dedicated systems - the design of the hardware and the software is dependent on each other. Therefore it is most important for the development and maintenance of future process control systems to define the basic terms used in connection with electronic components and their consequences. Such a common vocabulary will support the cooperation between hardware specialists, software specialists, application programmers and users.

This tutorial is mainly written as an introduction to the problems of microprocessors for the software-oriented computer scientists and application programmers. Therefore the reader should be familiar with the basics of computer science from the software point of view. A large number of references to more detailed publications completes the tutorial. Also many of the often used abbreviations are explained in short.

KEYWORDS. Computer hardware, computer software, computer programming, computer testing, distributed systems, micro, microprocessor, microprogramming, multiprocessing.

1. The dedication of process control systems

At the beginning of the automation of industrial processes only dedicated process controllers were used. These controllers were based on hardwired logic as well as analog techniques.

The first digital computers in this area have been installed as stand alone systems already in the early sixties. The necessary process interfaces, peripherals, A/D-, D/A-converters etc. were voluminous, susceptible to errors and expensive. Later on, process control computers were sometimes connected to large scale computers of general service centres. During this time the centralization strategy (DDC = direct digital control) was often discussed. For example the control algorithms realized in process controllers should have been removed to the computer. This idea was additionally supported at the beginning of the seventies by the new low-cost generation of minicomputers.

At the same time semiconductor components were not only integrated into computers but in process peripherals and interfaces as well. At first the user did not mention it, except its influence on prices and space requirements of hardware.

Today semiconductor components such as microprocessors and -controllers are most popular with manufacturers as well as users. The influence of such components on the architecture of process control systems is manifold:

- growing functional distribution to dedicated hardware modules (multi-processor systems, dedicated processors and controllers, etc.),
- growing specialization of basic components (integrated circuits), boards and subsystems,
- structure of system and application software,
- methods and tools (software and hardware) for the design, the programming and the test of microprocessor-based systems,

etc.

The main reason for arguing about the points mentioned above can easily be

explained: Microprocessors, micro-interfaces, memory-chips, etc. are basic electronic components of the present computer generation. General purpose computers are sold or rented including operating systems, compilers, hardware and software maintenance, and spare parts. The supplement micro in this context (micro-computer) covers very often marketing aspects only because it means that microprocessors and similar basic electronic components are used.

Important attributes of present process control applications are the

- requirements for the safety and reliability of hardware and software,
- the time behaviour (e.g. reaction time) as a decisive criteria for the faultless functioning of the system,
- the well-known type of problem(s) which should be solved by a specific system or subsystem,
- the lifetime of the software which is in most cases identical to the lifetime of the corresponding hardware system,
- the chance for the reusability of microprocessor-based (small) system components (e.g. boards) as an entirety of hardware and software,

etc.

These trends as well as the economical necessities of new application areas are encouraging the development and use of dedicated hardware systems.

Last but not least it must be clearly stated that microprocessors never will be cheaper successors of minicomputers. They are only basic electronic components with the help of which general purpose as well as dedicated computers, process controllers etc. can be constructed.

2. The terms microprocessor and microprogramming

As already mentioned the supplement micro is used in connection with very different things and meanings. In order to avoid misunderstandings of the arguments, given in this paper, it is necessary to clarify some of the most important technical terms in this area.

2.1 Microprocessor and microcomputer

Each computer system consists of hardware, system software and application software. The interface between hardware and software is mainly given by a set of machine instructions including the interrupt- as well as the I/O-functions.

The hardware contains in most cases also various levels. For this discussion the following partitioning of hardware will be used: electronic components boards and microprograms (fig. 1).

In the past the electronic components consisted mainly of relays, tubes, diodes and transistors. Today thousands of transistors including the additional passive electronic elements (resistors, capacitors, etc.) are integrated into one electronic element (IC = integrated circuit, chip). The chips are enclosed in small dual in-line packages (DIP; fig. 2) which take their name from the double, parallel row of leads (pins) that connect them to the printed circuit board (PCB). These basic components are called microprocessors, micro-interfaces, microcontrollers, semiconductor-memory etc. It is now possible to construct a complete computer out of a few "chips" on one single PCB (fig. 3). The performance and features of PCB's depend on the chips and the way they are connected to each other on the board. This holds true even if the same "CPU-chip" is used which defines the instruction set. The interrupt- and the I/O-structure is influenced by this board-architecture as well.

The architecture of a general-purpose computer includes the instruction set just as well as the interrupt structure, the I/O-facilities etc. All these features have consequences on most system software modules like operating systems, compilers and utilities. General-purpose computers are - as already mentioned - based on microprocessors and microcontrollers. Peripherals or control units also contain such electronic elements. Therefore from the technical point of view it is senseless to use the term microcomputer or to discuss minicomputers vs. microcomputers. The microprocessor is an electronic component and not a very low-priced replacement for a minicomputer. Nevertheless it is possible today to ease and to cheapen the development of computers which has also been proven by many hobbyists (shops, magazines and private organizations for personal computers (1,2)).

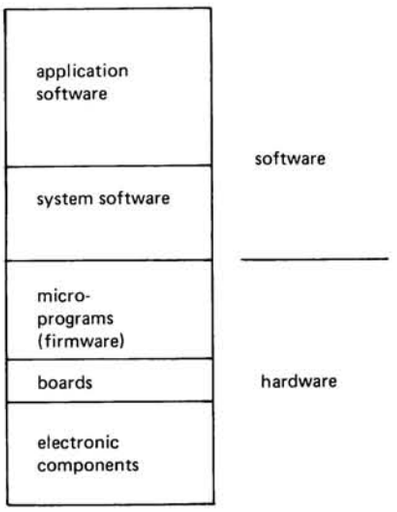

Fig. 1: Layers of a computer

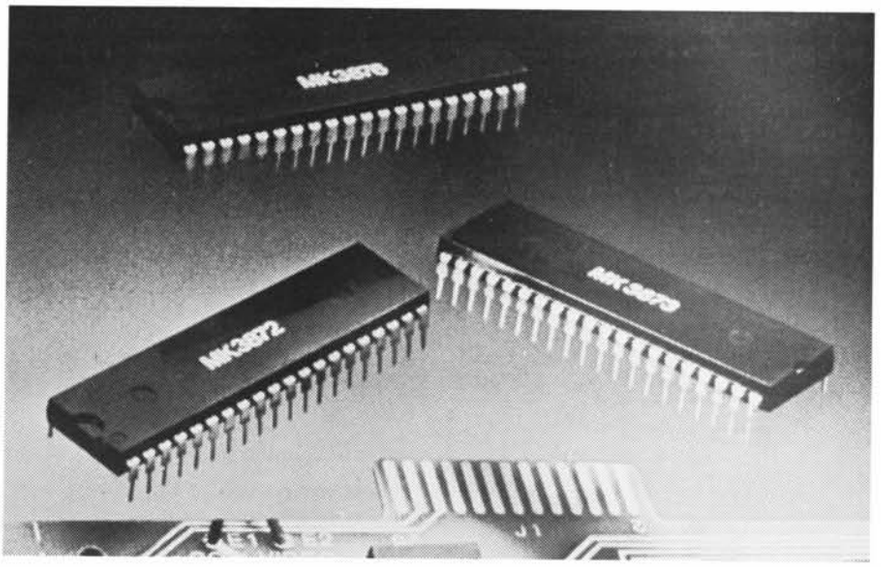

Fig. 2: Dual in-line package

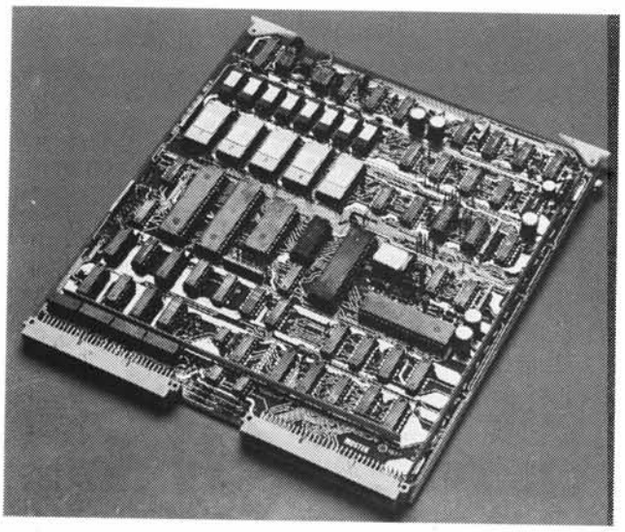

Fig. 3: Printed circuit board (Mostek SDB 80 system development board)

2.2 Classification of microprocessors

A steadily growing variety of chips is offered by semiconductor manufacturers. In order to be able to recognize the importance and consequences of the different types of microprocessors on applications, a classification will be introduced:

class 1 programmable microprocessors, and
class 2 microprogrammable microprocessors.

Programmable microprocessors are provided with a fixed set of machine instructions. Most instructions contain one operation code and one or two addresses. Therefore the addressing modes, address space, number and features of registers are also fixed.

The class 1 microprocessors are subdivided in

1-bit, 4-bit, 8-bit, 12-bit and 16-bit microprocessors.

The supplement n-bit means that the arithmetic-and-logical unit (ALU) of the central processing unit (CPU) is mainly processing operands with a length of n bits (except address operations).

The class 2 (microprogrammable) microprocessors are not provided with machine instructions. Before using such microprocessors an instruction set must first be defined by developing microprograms (see chapter 2.3).

The most important group of the class 2 microprocessors are the slice-processors. These are 2-bit or 4-bit wide "slices" of an ALU (e.g. 2-bit slices: Intel 3000-family; 4-bit slices: AMD 2900, Motorola 10800, Texas Instruments SN74S481). By joining together several slices a powerful ALU can be constructed. In the case of using four 4-bit slices, operands with a length of 16 bits can be processed in parallel (fig. 4). A survey on slice processors is given in (3).

2.3 Micro-instructions and micro-programming

A computer consists of hardware and software. The "interface" between both parts are the machine instructions. They can be implemented by hardwired logic (fig. 5a). Today this method is only used for very fast computers or computer components.

Another method (proposed by M.V. Wilkes (4)

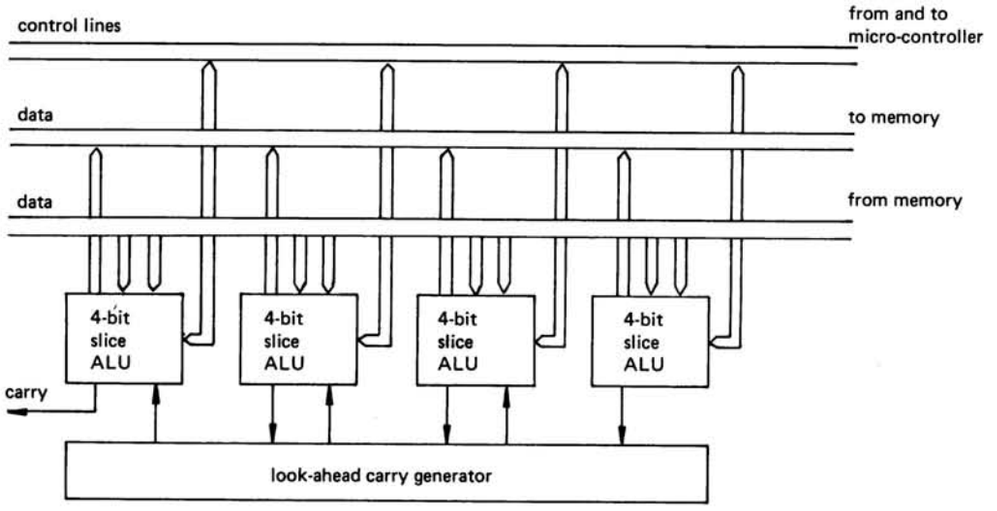

Fig. 4: ALU constructed with slice processors

already in 1951) divides the hardware itself in two parts. The lower layer contains a fixed (hardwired) net of basic functions (mainly gates). This network is controlled by so-called micro-instructions which are existing out of many short fields (often only one bit long). These fields are directly controlling gates (fig. 6). They are working independent of each other. Therefore a high degree of parallelism is typical for this lower layer. Other characteristics of micro-instructions are:

- Most micro-instructions do not have addresses (except instruction-addresses).
- Micro-instructions are long (40 to 120 bits) and their length is often not a multiple of 8.

Fields of micro-instructions longer than one bit must be decoded down to the gate-level. This encoding of fields is done in order to shorten the length of the micro-instructions.

The machine instructions are implemented in such systems by microprograms (microroutines) which contain a sequence of micro-instructions. In this upper layer of hardware (so-called firmware, fig. 5b) the following features of the CPU are defined:

- formats and effects of machine instructions,
- addressing modes and address space,
- data pathes,
- interrupt-handling,

etc.

Microprograms are stored in fast read-only or read/write memories (control store). Such a control store can also be placed on the CPU-chip of a programmable microprocessor (class 1, e.g. TI 9900).

The development of microprograms (microprogramming) is sophisticated and very close to the hardware. The main reasons for this are the high degree of parallelism and the time dependence. Unfortunately the available methods and tools are extremely poor.

Firmware can be used to construct a family of computers with identical instruction

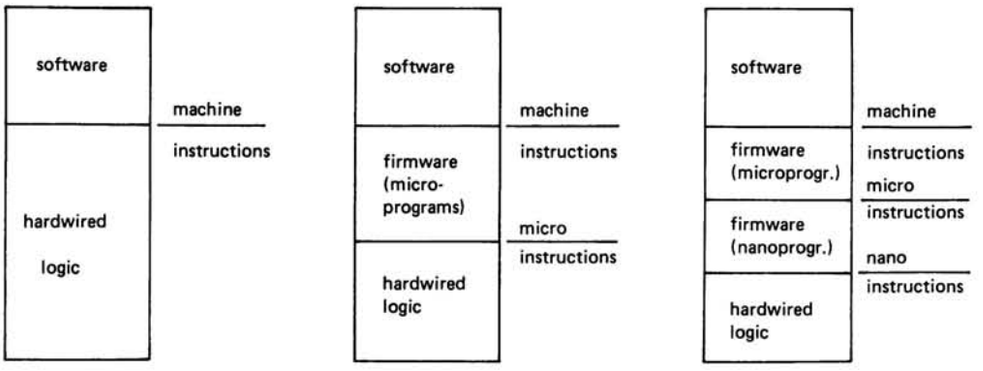

Fig. 5a Fig. 5b Fig. 5c

Fig. 5: Implementation of computers

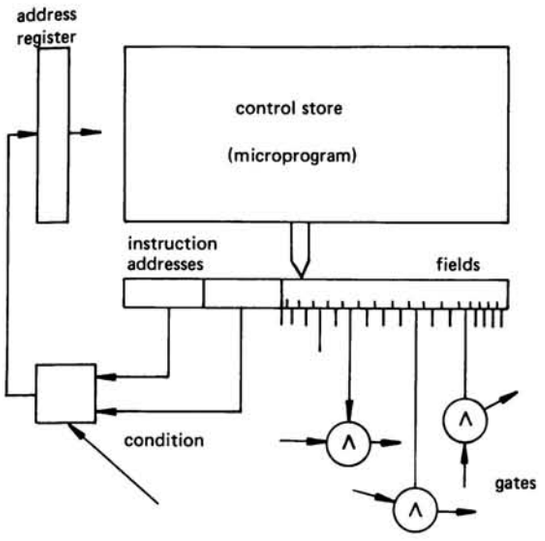

Fig. 6: Microprogram control

sets but very different technical implementations and different prices and performance. Nevertheless the software is compatible on the object program level (fig. 7). The development as well as the maintenance are done by the computer manufacturer. Without firmware it would not have been possible to construct series of machines which span such a wide spectrum of machine power (such as IBM /370 or PDP11).

The firmware of a computer is often not accessible to the user. Due to the availability of fast read/write memories this situation is changing. For some computers the manufacturers are offering the possibility of changing and extending their "standard" firmware (e.g. Burroughs 1700, PDP 11/60, Varian V73, HP 1000, Nanodata QM-1 etc). However, the micro-instructions of many of these "user-microprogrammable" computers are often similar to machine instructions with an operation code and operand-address(es). Therefore only a restricted access to gates, internal buses and registers is in this case possible. This is also the reason for dividing the firmware into two layers (fig. 5c). The corresponding "interfaces" between these layers are called nano-instructions (horizontal micro-instructions), micro-instructions (vertical micro-instructions) and machine instructions. A general introduction to user-microprogrammable computers is given for example in (5, 6).

Special-application firmware can be used to increase the level of "machine"-instructions (high-level-language machines; (7)) or to improve the performance of the computer with respect to the specific application. However, it must be taken into account that incompatibilities with the delivered operating systems and compilers can occur. Last but not least a new class of "programming" errors is created by it.

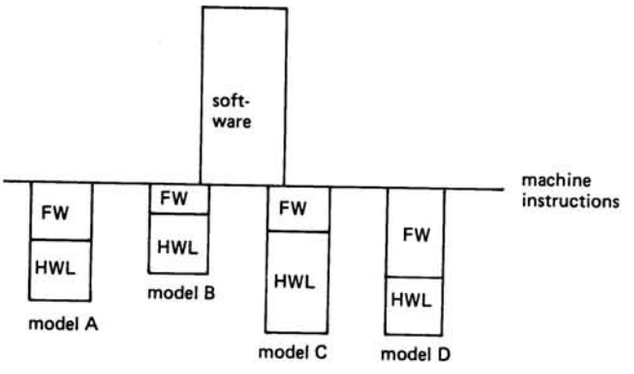

Fig. 7: Computer family

3. Function elements of microprocessor-based systems

A minimum system contains the central processing unit (CPU), I/O interfaces and memory which are connected by a bus (fig. 8). Additional elements like clocks, interrupt-, direct-memory-access- (DMA), D/A- and A/D-controller (DAC, ADC) supplement it. Due to the application-oriented requirements as well as the available technologies and production aspects these elements were put on single chips. The steadily growing number of basic circuits which can be integrated on a chip created the question which of the above mentioned basic function elements should be combined on a chip. In order not to mix the functional and the technological aspects, this chapter concentrates mainly on the function elements. The problems of the distribution of the function elements onto chips will be discussed in chapter 4.

As already mentioned in chapter 1 this tutorial concentrates on specific

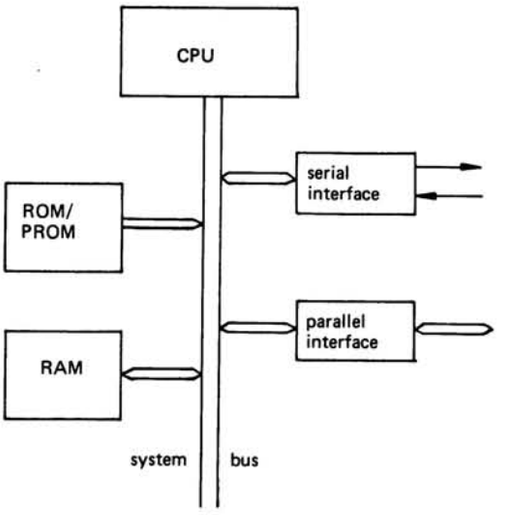

Fig. 8: Minimum system

microprocessor facts. Therefore the basics of computer science like data representation, addressing etc., which are well-known to programmers, will be referenced only shortly.

3.1 Central processing units

The architecture of a microprocessor CPU presents no peculiarities. It contains as usual the control unit (instruction decoding and execution), the ALU and the registers including the program counter and the status flags (fig. 9).

3.1.1 4-bit and 8-bit processors

One distinguishing feature of the many different CPU's is the length of the operands which can be handled by the ALU. The first available CPU's were able to handle 4-bit operands that means one binary-coded-decimal (BCD) digit by one machine-instruction. For longer operands an appropriate sequence of machine instructions must be given.

A short time after 4-bit systems (1971), 8-bit systems (1972) were already offered. The ALU of both systems in most cases was able to execute the following operations:

- addition and subtraction of unsigned integers (binary and BCD),
- logical operations (AND, OR, EXCLUSIVE OR, NOT) on 4-bit (resp. 8-bit) operands,
- compare, and
- several shift operations (with each instruction only one bit is shifted).

The carry-bit (which is handled by the ALU and special instructions like ADD and ADD WITH CARRY) eases operations on longer operands. Additionally a few operations on 16-bit operands are often available for address-handling. The multiplication and the division of integers must be executed by software (subroutines or macros). Obviously the same holds true for all floating point operations.

The status flags (zero, sign, carry, overflow, parity) are mainly set by the arithmetic and compare instructions.

Only a few 8-bit processors (e.g. Z80) offer besides basic logic- and load/store-instructions special (high-level) instructions for bit- and string-handling.

Usually the addresses consist of 16 bits. The address registers and the program counter are 16 bits wide as well. Therefore an address space up to 65 536 is directly available.

The machine instructions for program control like branch, conditional branch, subroutine call, subroutine return, increment and decrement (by one) etc. are available for most processors. Some of them offer special features like conditional subroutine call and return.

The organization of stacks is supported by most processors. The older ones have at their disposal a special stack memory integrated in the CPU (e.g. Intel 8008 or Signetics 2650). However, the overflow or underflow of the stack is not detected by the hardware. The other processors have a special (16-bit) stackpointer. The stack itself is placed in the memory by loading

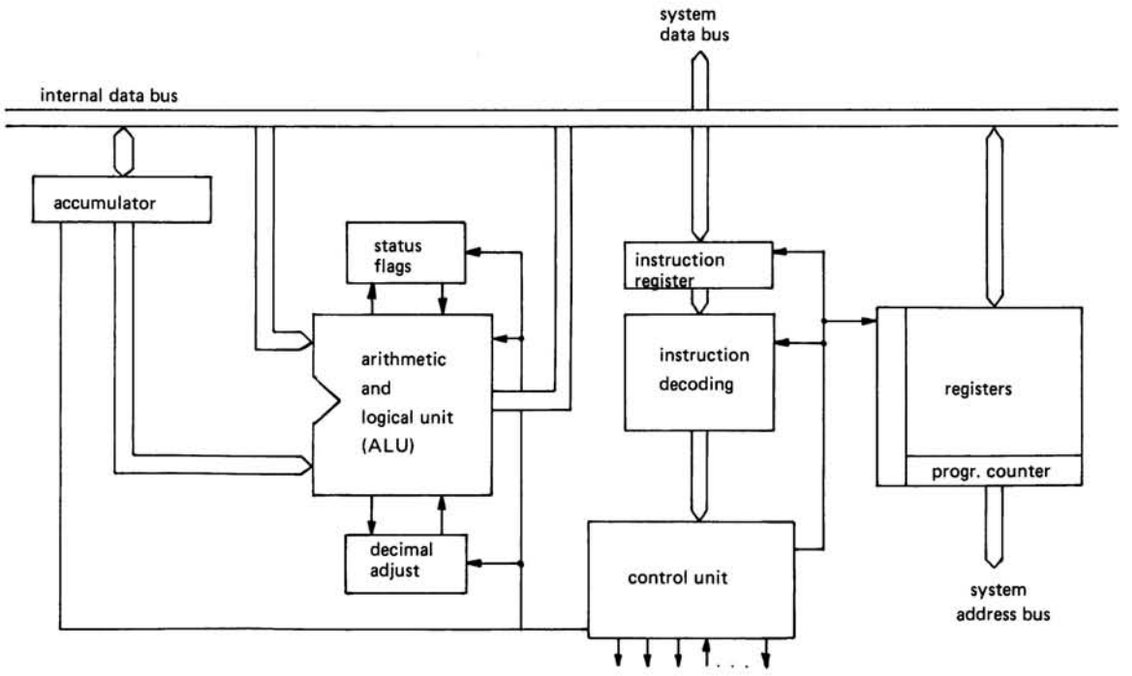

Fig. 9: Architecture of a CPU

the stackpointer. The stack for instance can be used for

- subroutine organization (subroutine call and subroutine return),
- storing of intermediate results, and
- saving of registers in the case of interrupts, etc. (PUSH and POP).

The stackpointer can be additionally handled by load-, store- and compare-instructions.

The number as well as the different kinds of registers are one of the main characteristics of the different types of CPU's. All processors are accumulator-oriented. Therefore at least one accumulator is available. Some processors have an accu-extension (e.g. National Semiconductor SC/MP) or a second accumulator (e.g. Motorola 6800) available.

The "arithmetic" registers (4-bits resp. 8-bits) are mainly used for intermediate results. For example the second operand of an arithmetic or logical operation can be taken from a register. Some processors allow the use of arithmetic registers similar to an accumulator for certain instructions.

The other registers with a length of 16 bits are mainly used for addressing and indexing. Sometimes also pairs of arithmetic registers can optionally be used as address registers. These registers can be handled with special instructions like load, store, increment and decrement (by one), exchange etc. which are necessary for address-handling.

The number of instructions vary from round about 10 to 150. Most 8-bit processors have approximately 70 instructions. This information given in data sheets of the processor must be checked carefully because the counting of instructions is somewhat difficult. The main reasons for this phenomenon are:

- the great number of addressing modes including different address lengths in the instructions (1 or 2 bytes), and
- the addressing of registers (including accumulators) is sometimes done in the operation code part of the instructions.

The length of instructions for most processors is 1, 2 and 3 bytes with an operation code of 1 byte. Remarkable is the great number of 1-byte instructions which have no visible operand address (inherent, implied addressing, register address integrated in the operation code).

Surveys on the available processors are regularly published by technical magazines (8,9). But in contrast to the past, it must be taken into account, that often the number and the performance of the instructions of the CPU is not the decisive point!

3.1.2 16-bit processors

Compared with the great number of 8-bit processors only a few 16-bit processors are available today (8, 10, 11, 12). The main differences are:

- a 16-bit integer arithmetic including multiply and divide is available, and
- the extension of the address-space to a length of 20 or 24 bits is often supported.

From the instructions' point of view this makes 16-bit processors much more similar as compared to traditional minicomputers. As a consequence of this some instructions need 4 bytes instead of 3 bytes. Especially for numerical-oriented applications the performance of 16-bit processors is higher and the corresponding programs are smaller.

In the past the 4-bit systems were superseded in most cases by 8-bit systems if new dedicated systems were designed. The same trend is often predicted for 8-bit and 16-bit systems. This certainly will hold true for general purpose computers like small business computers, personal computers etc. but not for great parts of dedicated systems. Many of these rather small applications are mainly operating on bits and characters except for addresses. Therefore they take no significant advantage of the special features of 16-bit processors. This problem will be also discussed in chapter 4.

A special case exists when 16-bit processors are discussed which offer an identical set of machine instructions as well-known minicomputers like PDP 11 and LSI 11 or Nova and micro Nova. A similar strategy was choosen by Texas Instruments. The family of the 990-computers is based on the TMS 9900 microprocessor family.

3.1.3 12-bit processors

A class of 12-bit minicomputers was developed in the past for process control applications. Due to the available measuring precision, 12 bits are enough for most variables. The PDP 8-family is a very good example. In order to replace today's comparitively expensive hardware without changing or reprogramming the existent software, compatible processors were developed. An example is the IM 6100 (13) which emulates the PDP 8.

A 12-bit processor was also developed by Toshiba (T3190). However, there will be presumably no realistic chance for 12-bit processors in the future because more powerful 16-bit processors are already available.

3.1.4 1-bit processors

The 1-bit processors have a different architecture as well as application area.

The processor only handles 1-bit operands which means logic operations on boolean variables. Instructions for program control like looping and conditional execution are also available.

The application area of such processors is mainly the replacement of hardwired logic in decision-oriented tasks. These processors are powerful elements in the construction of programmable logic controllers. The Motorola MC 14500 (14) was extremely successful during the last years.

3.2 Semiconductor memories

Memories in computers are used to store programs as well as data. An important feature of "program-controlled computers" in the very beginning (K. Zuse) was the possibility to store an instruction or data in a word of the memory and to change instructions (operation code or address) through the program itself. From the present software technology point of view, the second feature is especially bad. On the other hand this was the only way to implement vector-operations with a loop because machines like the IBM 650 had no indirect or indexed addressing. Therefore the address parts of the corresponding instructions must be incremented themselves. This example demonstrates the (in this case bad) influence of the hardware structure on the software. Semiconductor memories have an effect on the software as well.

Semiconductor memories are classified in two different types:

- read/write memories (RWM) which today are called random-access memories or read and modify (RAM), and
- read-only memories (ROM).

The RAM is used as a general working storage which can contain data as well as programs. Therefore data can be stored and read, and programs can be loaded for instance from disks and executed.

The main difference of RAM's to past memories is more a technical one. RAM's are so-called volatile memories. This means that the content of the memory will be destroyed in the case of a power breakdown. Therefore additional hardware like a battery back-up system is necessary which supports the RAM only during the power breakdown and which enables the processor later on to execute an automatic correct restart of the system. On the other hand a read operation does not destroy the memory content. Therefore an internal rewrite after read is not necessary.

Furthermore, the RAM's can be subdivided in static and dynamic memories. Static RAM's (based on flip-flop circuits) keep their content as long as the necessary power does not fail. Dynamic RAM's (based on capacitors) are permanently loosing their content due to the leakage current of the capacitors. Therefore they must be refreshed which is automatically done by hardware. However, this alternative influences only the hardware but not the software design.

The content of read-only memories can - during the execution of the program - just be read. Therefore it is only possible to store programs and program constants. They keep their content even if the power is switched off. Using memory-banking techniques, programs can be permanently stored in read-only memories ("resident" programs) instead of loading them from an external storage like a disk.

The read-only memories can be subdivided into four main-groups:

- ROM (read-only memories),
- PROM (programmable ROM),
- EPROM (erasable PROM), also called REPROM (reprogrammable ROM), and
- EAROM (electrically alterable ROM).

The content of ROM's is fixed in a mask which is later used for the production of the chips. Therefore the content of a ROM can never be changed. In the case of an error a new mask must be designed and new chips must be produced.

All other read-only memories can be "loaded" by the customer using special hardware. PROM's can only be loaded once. EPROM's can be erased by ultraviolet light (fig. 10) and afterwards loaded again. For this process a special hardware (PROM-programmer) is necessary.

The content of an EAROM can be changed in the target system during program execution if the necessary hardware is available. The write operation compared with the read operation is 50 to 1000 times slower. The EAROM's belong to the RMM's (read mostly memories) which are to be located between read-write and read-only memories (15).

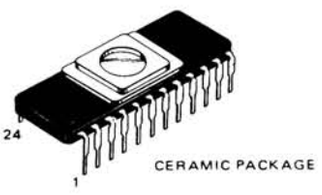

Fig. 10: Memory dual in-line package (EPROM)

In general for all semiconductor-memories address decoding and selection of the required memory position is integrated into the memory chips (see chapter 4).

The differences between read-write and read-only memories influence the software. For instance the program (instructions and program constants) and the data must be separated into two different types of memories. Another consequence is seen during maintenance. If programs are stored on magnetic tapes, disks, etc. corrections or modifications of programs are sent to the user on paper tape, punched cards or magnetic tapes. The integration can be done by the user himself. On the other hand if programs are stored on read-only memories, new chips must be manufactured (ROM) or burned in (PROM, EPROM). Even the replacement of chips cannot be done by the user. Therefore the elimination of errors during maintenance is extremely expensive. This fact strengthens the requirement to develop error-free programs.

A survey on the present state of semiconductor memories is given for example in (16, 17).

Finally it must be mentioned that new types of memories like bubble memories or CCD's will compete with external memories like floppy disks (18, 19, 20).

3.3 Interfaces

The elements discussed in 3.1 and 3.2 make it possible to construct minimum systems (fig. 8). Only one type of element is additionally necessary which controls the input/output operations.

The interface elements should allow the communication with

- data processing peripheral devices like teletype writers (TTY), paper tape readers, CRT's (cathode ray tube), printers, magnetic tape devices etc.,
- process control devices like process controllers,
- teleprocessing-lines (modems), and
- other computer-based components.

In general it must be taken into account that nowadays most external devices are based on microprocessors. Therefore most of the communication between the "computer" and peripheral devices is implemented as a computer/computer connection from the hardware point of view. The device-dependent operations are implemented as software partially in the "computer" and in the device. For instance plausibility checks of measured values can be executed in process controllers or text editing can be implemented in terminals.

Two basic types of interfaces are available:

- serial interfaces (communication interface, USART = universal synchronous /asynchronous receiver/transmitter, ACIA = asynchronous communication interface adapter, SSDA = synchronous serial data adapter etc.), and
- parallel interfaces (PIO = parallel I/O interface, PIA = peripheral interface adapter, PPI = programmable peripheral interface etc.).

These terms and abbreviations are manufacturer specific ones.

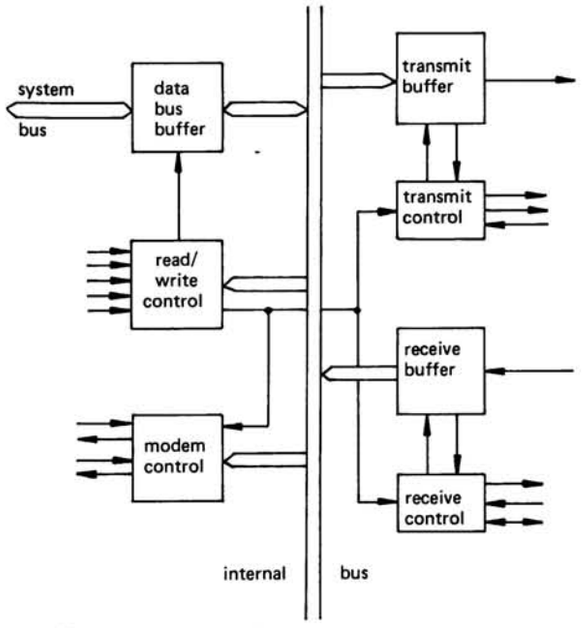

Fig. 11: Serial interface

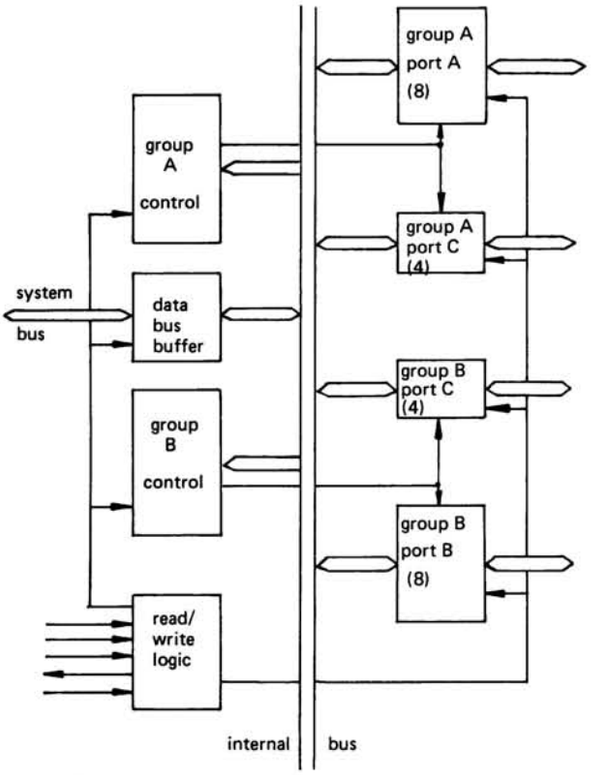

Fig. 12: Parallel interface

Both types of interfaces can be used for a wide range of applications. In order to minimize the number of different chips, the speed of the data stream (Baud rate) or the mode of operation (synchronous or asynchronous) etc. can be choosen during execution by sending a control word from the CPU to the interface. Such flexible interfaces are called programmable interfaces. The term "programmable" in this case means only that the transfer speed, the mode of operation etc. can be selected by software in sending control words from the CPU to the interface!

The general structure of a serial interface (e.g. Intel 8251 or Motorola MC 6850) is given in fig. 11. On the left hand side a data bus buffer interfaces this element to the CPU. This buffer is normally 8 bits wide. It gets control words or data (to be output) from the CPU or readies data (which were received) for the CPU. The transfer of information between CPU (register or accumulator) on the lowest level is executed by machine instructions (e.g. IN, OUT). In order to control these transfer operations some control signals are necessary.

On the right hand side a separate transmit and receive unit (buffer and controller; fig. 11) send and receive the information. These controllers

- control the data transfer,
- check the data transfer (parity, overrun, framing etc.),
- convert the data from parallel to serial (output) or serial to parallel (input).

Both parts of the interface are connected by an internal 8-bit wide bus.

The general structure of a parallel interface (e.g. Intel 8255 or Motorola MC 6820) is given in figure 12. The left hand side looks very similar to the serial interface. On the right hand side I/O ports are shown which can be applied in a very flexible way. For example they can be connected with a floppy disk controller or separate process signals.

Several parallel interfaces offer different modes of operation such as:

- basic input/output,
- strobed input/output, and
- bi-directional bus.

In all modes 8 lines are used for data. In the case of basic I/O no control signals are available. The I/O operation must be completely controlled (including timing) by the CPU.

The strobed I/O mode offers three control lines for handshaking. The input device for instance can send a request signal to the interface using one of the control lines. Strobed I/O is used for example to communicate with printers, A/D converters, machine tool controllers, etc.

The bi-directional bus mode needs 5 control lines. It is, for instance, used to communicate with floppy disks or with other processors (multiprocessor systems).

As already mentioned at the beginning of this chapter most I/O tasks can be solved with the help of the two types of interfaces. Nevertheless simple elements like ports or more sophisticated elements like CRT controllers are available. The low level elements (e.g. ports, address-decoders, clock generators, bus drivers) do not influence the software and the system as a whole. Therefore they will not be discussed in this paper. The high level elements will be discussed in chapter 3.5.

3.4 Buses

The function elements need means for the transfer of data and the exchange of control signals. A so-called system bus connects for example the CPU and the memory, the CPU and the interfaces (fig. 8). System buses of most 8-bit systems consist of 8 data, 16 address and some control lines. The bus is controlled by a functional element called bus driver which is a separate chip or integrated into the CPU. System buses are defined by the semiconductor manufacturers. Therefore no standard is available.

Some function elements like CPU's and interfaces use an internal bus (on the chip) for the communication inside these elements. These buses are only interesting during the development of the circuits which will be integrated onto one chip.

The communication between two processors can be done using serial or parallel interfaces. A more general solution for multiprocessor systems is possible by an external bus (21, 22, 23). This bus also should support for example a shared memory which allows access from different processors. A standard for external buses compared with system buses is much more important because often very different processors are used in multiprocessor systems. But up to now, besides the manufacturer's conventions, only a few standard proposals are submitted:

- the IEEE-488 bus ("instrumental" bus, originally developed by Hewlett-Packard (24, 25)),
- the PDV bus and SIR (proposed by the TC 7 of Purdue Europe (23, 26, 27, 28)),
- the Intel Multibus (developed by Intel (29)),
- the STD-Z80 bus (developed by Mostek (30) and also used e.g. by Prolog), and
- the S100 bus (widely used for personal computers (31)).

Although the standardization is urgent it will still take some time.

3.5 Special function elements

The function elements discussed up to now are well known basic elements of computers. Dedicated systems constructed from them, can be compared with general purpose computers with a medium performance (execution time of machine instructions and size of memory).

Additionally special function elements are available which should fulfill one or both of the following objectives:

- improvement of the performance of dedicated systems (arithmetic operations, data transfer etc.),
- replacement of software routines or software modules, and
- replacement of discrete logic on a board by "chips" (timer, D/A and A/D converters etc.).

These elements are divided into three groups: basic, arithmetic and controller elements. However, the function elements which are only relevant for hardware design will not be mentioned.

3.5.1 Basic function elements

The basic function elements mainly complete the elements mentioned in the previous chapters for real time applications. They are replacing discrete logic or software. Two simple examples are timers and counters (e.g. Intel 8253). A timer (fig. 13) contains a register which is automatically decremented, controlled by a crystal or a similar source. This register can be loaded from the processor. If the counter is decremented to zero an interrupt is sent out. The counter works in a similar manner. Only the decrementing of the register is initialized by external events. Such counters are for example able to reduce the number of interrupts which must be handled by the CPU.

Another much more important basic function element is the so-called direct memory access (DMA (32); e.g. Intel 8257, Rockwell DMAC). In the case of large data rates which must be input or output by one or more interfaces of a system the organization of this transfer is somewhat slow and awkward. The reason for this fact is that the data must be passed through the accumulator or a register of the CPU. Therefore each byte must be handled by the processor. If several interfaces are simultaneously working, a considerable overhead is caused which significantly reduces the performance of the system. In this case the performance can substantially be improved by using a DMA element which directly controls the transfer of data from an external device (interface) to the memory or vice versa (fig. 14). This transfer must be only started by the CPU in giving the identification of the interface, the start address in the main memory (I/O area) and the number of bytes to be transferred. The DMA element transfers the data in a cycle-stealing mode. Also several DMA's can work in parallel. At the end of a DMA-operation an interrupt is sent from the DMA element to the CPU.

Finally, as an example for basic function elements especially necessary in process control applications, the analog/digital

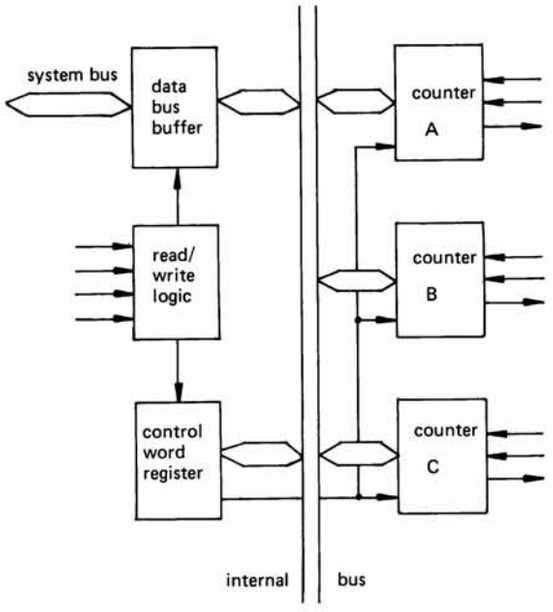

Fig. 13: Timer/counter

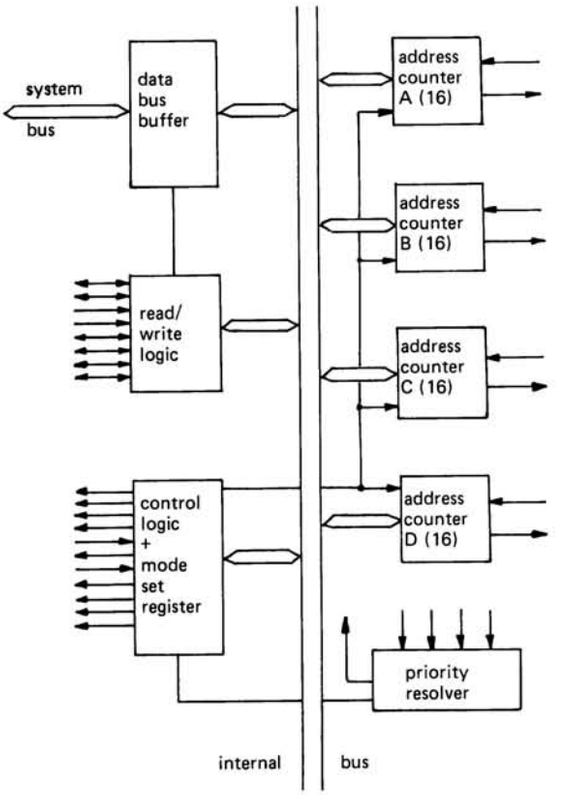

Fig. 14: DMA-controller

and the digital/analog converters (ADC and DAC; (33, 34)) should be mentioned. These elements are now also available as single chips.

3.5.2 Arithmetic elements

In particular the performance of 4-bit and 8-bit processors - with respect to numerical applications - is often insufficient (see chapter 3.1). Even 16-bit processors do not have built-in floating point features available. One way out of this problem is the application of dedicated arithmetic processors based on fast slice processors which are offered in the form of complete boards. Another possibility is given by arithmetic elements (35) like fast multipliers (e.g. MMI 67558, TRW MPY-8) or floating point elements (chips). These function elements are similar to specialized, efficient processors. Also elements for Fourier analysis are already available.

The arithmetic elements are connected to the system bus. Therefore the transfer of data can be controlled by DMA's.

3.5.3 Controller elements

During the last two years a new kind of specialized, efficient elements which take over the function of complete microprocessor-based systems (hardware and software) or discrete logic have been created. Such elements should be called controller elements. The following are well-known examples:

- the controlling of a CRT-display (CRT-controller; (36, 37, 38)),
- the controlling of a floppy disk drive (e.g. National Semiconductor INS 1771; (39)),
- the controlling of a keyboard and a display (7-segment display; e.g. Intel 8279, MTX-A1; (44)), and
- the controlling of a teleprocessing line in SDLC mode (synchronous data link control; e.g. Intel 8273; (40, 41)).

CRT-controllers (National DP8350, Motorola MC6845, Intel 8275 etc.) support in most cases the following features:

- character generation using an additional read-only memory and optionally blinking, underlining, reverse video and highlighting,
- simple graphic elements to construct tables at the screen,
- cursor and lightpen support, and
- refreshing of the display.

Using such an element, a complete controller can be realized with only a few additional chips which are handled by the CPU in a very simple way.

The other controller elements are similarly efficient and they simplify and cheapen the controlling of external devices (42, 43, 44, 45). Also the performance of the whole system is considerably influenced. Last but not least a certain kind of de-facto standards is supported using the economical outcome.

4. The distribution of function elements onto chips

At the beginning of microprocessors the only possibility was to put the CPU (or a great part of it), the interfaces and memory modules each on one chip. Additionally discrete logic was often necessary to complete a system.

Due to the steadily improving technology as well as production techniques a still growing number of circuits can be integrated onto one chip (table 1). Today often two or more (identical or different) function elements can be integrated on one chip. This leads to a great number of alternative possibilities. The following criterias are to be taken into account for the choice of the combination of function elements:

C1 the number of pins available on the DIP (dual-inline package) for the chip (alternatively 14, 16, 18, 20, 22, 24, 28, 40 and sometimes 64 pins),
C2 the number of chips which can be sold per year in order to be able to offer fair market prices (return of investment),
C3 the minimization of the number of different chips in a chip family, and
C4 the minimization of the number of chips of complete systems.

	1972	1974	1976	1978	1980	1984
logic gates density / mm^2	100	130	170	230	300	600
gate complexity (gate / chip)	750	1 500	3 000	8 000	19 000	45 000
chip space (mm^2)	12.2	18	21.4	33	45.5	60

Table 1: Trends of chip characteristics

Some side-effects of these criteria are important, e.g. C2 promotes the definition of de-facto standards and their compliance. Obviously, these criteria also contain some contradictive requirements like C2 and C4 (generalization vs. dedication).

First of all some examples of chips will be discussed. The capacity of memory chips is given as the number of bits or more precisely in the form

n * m,

where n is the number of units with a length of m bits. For instance a RAM of the type

1024 * 4

contains 4096 bits of memory where each of the 1024 4-bit groups can be addressed. Therefore the 20 pins are allocated as follows:

10 pins	address
4 pins	data input/output
2 pins	chip select
1 pin	write enable
1 pin	output disable
2 pins	power
20 pins	

Further examples are given in table 2.
The CPU is another example. The 40 pins for instance of the Intel 8085 are allocated as follows:

8 pins	address (output)
8 pins	multiplexed data/address (input/output)
9 pins	controlling of the buses (address latch enable, data bus status, read, write, ready, hold)
6 pins	interrupts and administration of interrupts
2 pins	serial I/O
2 pins	reset
3 pins	clock (input/output)
2 pins	power
40 pins	

Different approaches are applied especially for CPU's. One objective requires integrating of all basic function elements like interfaces and memories (ROM and RAM) onto the CPU-chip in order to get "one-chip computers" (46, 47, 48, 49). Examples are given in table 3.

Another objective is also used: design of flexible and powerful CPU's which allow the construction of a wide range of systems (size as well as type). Obviously, a complete family of chips for the other function elements is required. Some examples are given in table 4.

Going on to other basic function elements for DMA's, timers, counters or interrupt controllers, several identical elements are often integrated on one chip. The Intel 8257 contains 4 independent DMA controllers and the Intel 8253 three counters which can be used for example as event counters, real time clocks or rate generators.

A mixing of function elements is not only available for CPU's. Interfaces, for instance, are also integrated onto memory chips (Intel 8155/8156, 8355/8755). This allows the reduction of chips especially for small systems.

A very popular discussion is going on with the topic of 8-bit vs. 16-bit processors (see chapter 3.1). Without any doubt, 16-bit processors (which include the complete integer 16-bit arithmetic) are very interesting and better than the 8-bit processors for small general purpose computers or mainly numerical oriented systems. On the other hand, if dedicated systems are required which mainly handle bits and character strings, then 8-bit processors also offer many advantages which are mainly: sufficient instruction sets, flexible system structures, large and powerful families of chips and boards, hardware and software tools for system development (monitors, assemblers, compilers, simulators, debugging aids etc.), many second sources of the chips and boards, many alternative chip families (8).

memory type	vendor	model no.	size (Kbit)	organization	access time (ns)	number of pins
ROM	MMI	6282	8 192	1024 x 8	55	24
ROM	Mostek	MK 36 000	65 536	8192 x 8	350	24
PROM	Intel	2616	16 384	2048 x 8	300	24
EPROM	Intel	2716	16 384	2048 x 8	450	24
EPROM	TI	2532	32 768	4096 x 8	450	24
RAM	Mostek	MK 4007	256	256 x 1	450	16
RAM	Intel	2111	1 024	256 x 4	1 000	18
RAM	NS	2112	1 024	256 x 4	1 000	16
RAM	Motorola	MCM 6616	16 384	16384 x 1	300	16
RAM	Mostek	MK 4816	16 384	2048 x 8	120	28

Table 2: Memory chips

The usage of function elements like DMA's, interrupt controllers, counters and all controller elements demonstrate the considerably improved performance of dedicated systems which can be reached without changing the CPU or rather its instruction set. Another example is given by applications which are mainly operating on matrices: the usage of an 8-bit processor in connection with an array-processor. In this case the 8-bit processor concentrates on the management of the system while the array-processor takes over the matrix operations in a very fast manner (much faster than a 16-bit processor! Fig. 15).

Finally going back to the problem of distribution of function elements onto chips, another point must be taken into account. Due to the improved technology and production techniques some free space is left on the chips which can be used for additional circuits. One possibility is to design 16-bit processors, 32-bit processors, one-chip computers etc. For some applications this is really very interesting and important (from the technical as well as the marketing point of view). However, there are other possibilities too:

- integration of error checks and analysis for data handling and transfer,
- duplication of function elements or circuits which take over if others fail,
- duplication of critical function elements or circuits in order to improve the production rate of chips, and
- to speed up the function elements (which also needs space).

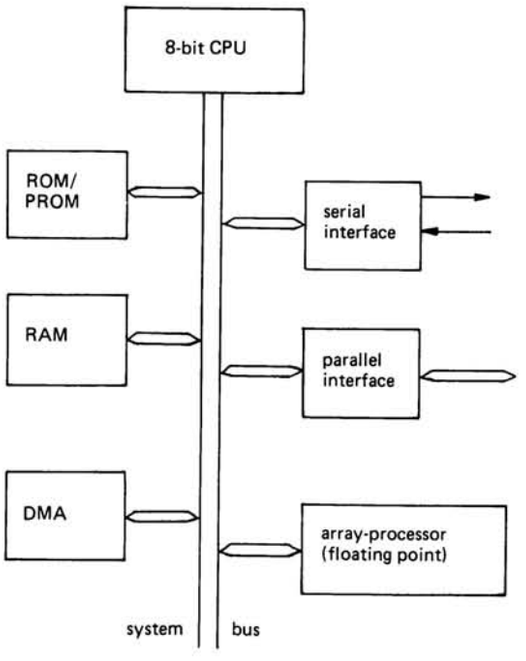

Fig. 15: 8-bit processor with an array-processor

One of the main problems seems to be the recognition of the differences between general purpose and dedicated systems and to analyze the consequences of the new semiconductor elements to the architecture of hardware and software.

5. Hardware development

As already mentioned in the introduction, the hardware will be discussed as long as the system architecture or the software is influenced. Therefore the problems of hardware development will be only roughly presented.

There are different levels available from which system development can start (fig. 16).

The lowest level consists of the design of special purpose circuits on a chip (custom IC's) or the modification, adaptation or extension of existent chips by changing their PLA (program logic array (50, 51)). This level should only be taken if a large number of identical chips is needed. This method is mainly used by semiconductor manufacturers. The floppy-disk and the SDLC controller from Intel (8271 and 8273) for example share the same architecture on the chip (52).

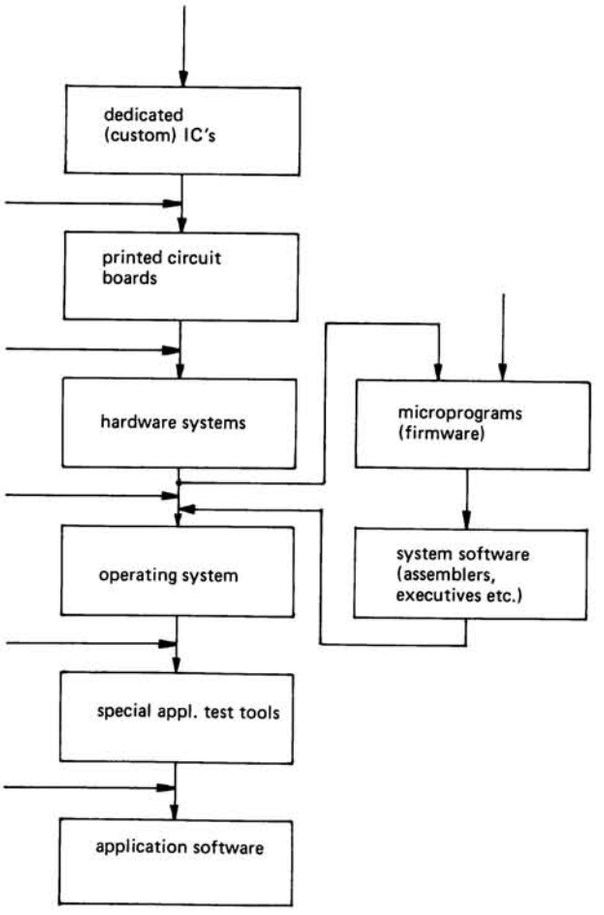

Fig. 16: Development of microprocessor based systems

Another possibility for high-speed or special applications offer slice processors (3) which allow, for instance, the construction of a 64-bit ALU. In this case standard chips can be used but the instruction set of the CPU must be defined by developing microprograms (firmware).

The next level uses available chips but designs a dedicated system architecture. On one hand additional adaptations to external devices or other components using discrete elements like transistors, diodes etc. must often be developed. On the other hand these dedicated boards can be combined with available "standard" boards.

It is often possible to start on the next level with a family of "single-board computers". Such a family consists of CPU-boards, memory boards, interface boards, analog boards (with ADC and DAC elements), memory/interface boards and power supplies (e.g. Intel SBC (53), Mostek SDB (54), Siemens SMP (55); fig. 17). If only a few (hardware) identical systems are required, then this method can save time and money.

Finally it is also possible to start with a complete system which, for instance, can be extended by such things as additional memory boards etc. In many cases such a system is not a general purpose or dedicated computer but rather a special purpose device like a CRT-display or a process controller. Such systems already contain software modules (ROM's, PROM's) which must be modified or extended.

Obviously the hardware should be developed by hardware specialists. Nevertheless a tremendous number of hobbyists is successfully implementing such systems. The hardware development is supported by tools like design kits, logic-state analyzers (more powerful replacement of oscilloscopes, (56)), in-circuit emulators (see chapter 7), development systems etc. The availability of such tools influences the selection of the type of processor or board which will be used for a system (57).

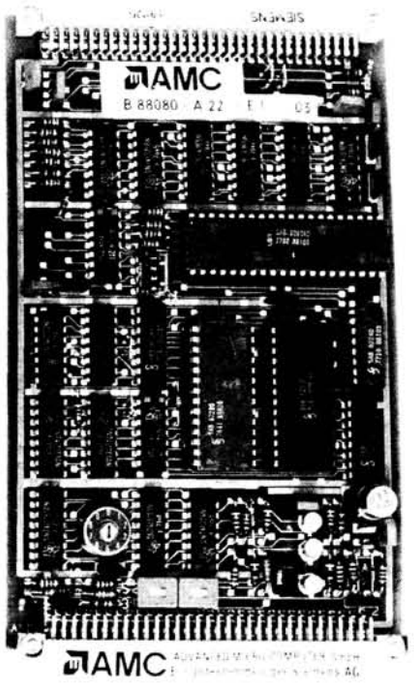

Fig. 17: Single-board computer (Siemens SMP 80)

Finally an additional criteria for the decision on which level the system development should be started comes from the area of maintenance. The availability of chips and boards as spare parts (stockkeeping) has a remarkable effect upon the calculation of the final price of the system.

6. Software tools

After mainly discussing hardware aspects, the problems of software development will be analyzed. In this part only the progammable processors (see chapter 2.2) will be considered and predominantly dedicated systems are taken into account.

6.1 Programming languages

Most programs for programmable microprocessors are written in assembler languages, which are available for all processors. There are two main reasons for the usage of the hardware dependent (or rather semiconductor manufacturer dependent) low-level languages:

- there are only a few high-level language compilers available with a high performance and test aids, and
- in most cases the dedicated (hardware) systems and the required reaction time lead to the necessity of hardware-oriented programming.

Alternative solutions of this problem will be discussed later on.

The assembler languages for microprocessors look similar to those of minicomputers. Difficulties mainly come from terms without uniformity for instance for the different addressing modes and confusing mnemonics of the operation codes:

JMP m means on the Motorola 6800: jump to the subroutine, starting at location m (branch and link), and on the Intel 8080 means: continue program execution with the instruction sequence starting at location m (branch).

Other examples are:

- the operation "exclusive or" is written as
  EOR (Motorola 6800)
  XRA (Intel 8080)

- and the "branch and link" of subroutines:
  JMP or BSR (Motorola 6800)
  CALL (Intel 8080)
  PI (Fairchild F8)
  BL (Rockwell PPS 8)
  BSXA (Signetics 2650)
  JSR (MOS Technology MCS 6500)
  etc.

Therefore the experienced programmer easily understands the architecture of a new microprocessor but during coding many errors can occur which are not so easy for him to analyze. Also the readability of assembler programs is additionally worsened by it. Some proposals (58, 59, 60, 61) have already been made in the past for general purpose assembler languages (or rather mnemonics) but the assemblers themselves and the necessary additional system software must also be available and its maintenance must be guaranteed. From the present point of view standardization on this level will have no realistic chance.

Comparing assembler programs of mini-computers like PDP11 and 8-bit processors it is easy to see that the programs for the 8-bit processors need more assembler statements even if the object program is of similar length or sometimes shorter. However, some short instruction sequences are repeated many times. Consequently macroprocessors which are used in connection with assemblers (macro-assemblers) are a very helpful tool. They are offered for many processors. However, the usage of macro-assemblers is somewhat awkward if no macro-library organization (e.g. automatic fetch of macros from the library) is available.

Another way out of the problems of assembler mnemonics is the usage of system implementation languages (62). Such languages allow users to write hardware dependent programs in a high-level language style. One step in this direction was done for 8-bit processors from Intel by developing the language PL/M (63, 64). The syntax of PL/M is a very small subset of PL/I. Only two data types: 8-bit data (unsigned integer) and 16-bit addresses are available. The following example defines a procedure taken from a real-time executive. It gives an idea of what PL/M programs look like:

```
XENTRY: PROCEDURE (PRIORITY) PUBLIC;
/* A PROCESS ENTERS A CRITICAL SEQUENCE*/
DECLARE PRIORITY BYTE;

IF CP.CRITSECTION    = NOCRITSECTION
   THEN CP.CRITSECTION = PRIORITY;
   ELSE HALT;
NEXTPTR = PRIOTBL (PRIORITY);
*/ BLOCK OTHER PROCESS */
NEXT.STATE = NEXTSTATE OR BLOCKED;
ENABLE;
END XENTRY;
```

PL/M does not contain any language elements for input/output, real-time functions, tasking etc. However, existing assembler routines (real-time executives, I/O drivers etc.) can easily be integrated as PL/M-procedures.

The absence of such "operating system"-features as language elements of PL/M seems to be a disadvantage from the application programmer's point of view. On the other hand it must be taken into account that dedicated systems often need dedicated (and not general purpose) interrupt handler, memory banking, I/O interfaces, controller elements etc. This makes it (economically) impossible to directly fix these hardware-architecture-dependent features to the programming language which cause certainly a large overhead (memory space and slow reaction time).

The efficiency of the object program generated by the PL/M compiler, running on the development system Intellec II (65), is quite good but critical inefficient instruction sequences should be replaced by assembler statements.

The basic concept of PL/M has been adapted by some other manufacturers like Motorola (MPL), Signetics (PLuS), Zilog (PL/Z-SYS)). These languages are more or less derivatives of PL/M (66, 67).

Analyzing the experience with PL/M and its derivatives it must be stated that such programming languages are, in most cases, much better than assembler languages. Nevertheless the definition, implementation and usage of an implementation language with a unique syntax for different processors is very important.

For several widely-used processors BASIC and subsets of FORTRAN IV and for a growing number of processors also PASCAL (68) is offered. Extensions of these languages for real-time applications have been proposed and are already used (RT-BASIC (69), RT-FORTRAN (70) and Concurrent PASCAL (71)). However, the corresponding fact, as discussed for PL/M, can be recognized for these high-level languages which include for instance I/O-functions. The compiler and the run time system which support these features are dependent of the hardware architecture. Therefore these languages and compilers are restricted to "standard" architectures if these language features are used. Obviously (as done in PL/M) assembler routines can be integrated into high-level language programs for most processors.

It should be also mentioned that these high-level languages are mainly designed for numerical applications and not for bit and string manipulation. On the other hand most applications of dedicated microprocessor-based systems concentrate on bit- and string-handling! Therefore they are often

not very suitable for the programming of dedicated systems.

The application of programmable microprocessors in small business computers, personal computers etc. are leading to the implementation of other languages like subsets of COBOL (Zilog, Mostek) and PL/I (Mostek; under development).

Up to now mainly the programming of dedicated systems was discussed which were designed for a specific type of application. Finally a special case will be discussed which is of growing importance. Some dedicated systems are developed which include system software as well as the basic application modules (delivered in PROM's or ROM's). Examples are terminals and process controllers. In most cases such systems must be adapted to its environment by extending its software. Due to the great variety of processors, interfaces, hardware architectures etc. the user is often not able to do this job or it is too dangerous for him (sophisticated errors). Also the maintenance of hardware and software must be taken into account. A way out of this situation can be found by using fill-in-the-blanks languages (fill-in-the-form) and the corresponding program generators (72). The language ACCOL can be taken as an example. It was designed and implemented from the aluminum company ACCO for a process controller based on the 8080 developed by the same company (73). In the case of fill-in-the-blanks languages the user is "programming" by only answering application-oriented questions. The answers are used to generate the object program (main program) which makes an extensive use of the already available program modules and program and data structures. This method prevents errors (in the sense of computer-science) which very often are produced in high-level-language programming. However, a remarkable investment (time and manpower) for such tools is still necesssary today.

6.2 Assemblers and compilers

The software for large scale and minicomputers is developed on a target computer or the same type of computer. Also the same operating system is used. In the case of dedicated systems this method is often not applicable. The target system has not available the necessary hardware (memory, external memory, I/O devices etc.), and the real-time executive is not able to support assemblers or compilers, test tools etc. Therefore host computers are used which are based either on the same processor (direct techniques) or on another processor (cross-techniques).

If the same processor is used, so-called development systems (see chapter 7) are available. The assembler and compilers generate object programs which can also be executed in most cases on such development systems.

The assemblers and also some small compilers are sometimes offered in two different versions. In the first case the assembler or compiler is stored on a floppy disk or other external storage and loaded for execution into the main memory. In the other case it is stored on ROM's or PROM's (resident assembler or compiler). Because of memory-banking no extra address space must in this case be reserved.

If host computers are used with another type of processor, cross-assemblers or cross-compilers must be available. In this case object programs for the target processor are generated on the host system (fig. 18). These object programs can be loaded and executed either on the target system (using for instance paper tapes) or on the host computer with the help of a simulator (see chapter 7). The development system can also act as a host computer (e.g. the Tektronix development system for 8080, 8085, 6800, etc. (74) or the Mostek development system based on Z80 for the Fairchild F8 (75)). In the case of

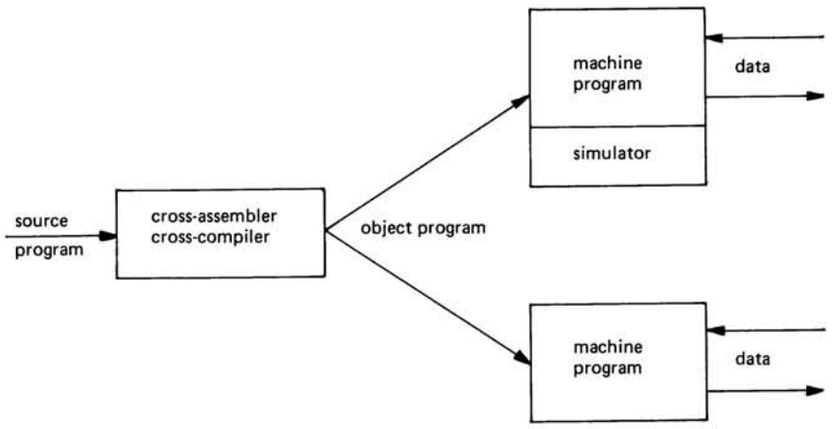

Fig. 18: Cross-techniques

cross-assemblers and cross-compilers - working on development systems - it must be carefully checked whether the language syntax and semantics are identical to the primary system. Otherwise the incompatibilities may cause difficulties for instance during maintenance.

Most cross-assemblers and cross-compilers (especially for general purpose computers as hosts) are written in FORTRAN IV. Therefore they can easily be adapted to many host computers.

Furthermore some other points are important in the sense of the practability of such tools. Floppy disks or similar external memories are not a must (source programs, intermediate results, listings, object programs etc.) but they considerably speed up the compile time. Nevertheless the compile time for large programs is often remarkable. This has an unfavourable effect in particular during system test and maintenance. Therefore it should be possible to assemble or compile source modules separately. As result a linkage editor is also required.

6.3 Operating systems

For dedicated systems it is necessary to make a distinction between operating systems for program development and program execution. Operating systems for program development are designed and implemented for development systems (see chapter 7). Today they are (floppy) disk operating systems including assembler, compilers, linkage editor, text editor, data management, utilities, debugging aids etc.

As support for the execution of object programs only kernels of real-time operating systems (monitors, real-time executives) are offered by manufacturers. They need between 0,5 and 2 Kbyte of main memory, delivered on a chip (ROM or PROM). The source of these executives is in most cases available. However, the extensions for I/O drivers, interrupt handling inter-processor communication etc. must be designed, implemented and tested by the user!

There are also several kernels for single- and multiprocessor-systems designed and published by institutes or users (76, 77, 78). Many of them are based on Wirth's Modula (79). Nevertheless the design and implementation of a dedicated operating system (extension or modification) is still a critical factor of development projects. The consequences of maintenance aspects must especially be taken into account.

Therefore methods and tools which allow a program-controlled generation of dedicated real-time operating systems is very important. A first step in this direction was done by Intel developing RMX/80 (80, 81) which also supports multiprocessor systems. It includes data management features for floppy disks and a free space manager. The source of RMX/80 (mainly programmed in PL/M) is not available for the user. Nevertheless some disadvantages should be mentioned. The responsibility of the application programmer, in the case of multiprocessor systems, is still high (possibility of programming errors). Furthermore the user is forced to sign a contract that RMX/80 will only used for systems based on Intel SBC (single board computer). Beside this contractual reason there are also some technical reasons which hinder the usage of RMX/80 on other 8080- or 8085 based systems. Even on the Siemens SMP (which is a version of Intel's SBC on Euro-cards) RMX/80 could only be used if the boards are modified.

The development of methods and tools for the hardware- and manufacturer-independent dedicated operating systems (operating systems generators) is very important (82, 83). Multiprocessor systems should be paid special attention to.

7. Programming and software testing

The problems of programming and software testing will mainly be discussed with respect to direct techniques (see chapter 6) and the application of development systems.

Development systems are based on microprocessors (8-bit processors in most cases) and should contain the following hardware components:

- 32 to 64 Kbyte read/write memory (systems with memory banking can be extended up to several Mbyte),
- complete interfaces for teletype, CRT, paper tape reader and puncher, printer and floppy disks,
- 2 to 4 disk drives, and
- additional hardware tools for software tests.

The corresponding system software exists of a (floppy) disk operating system, an assembler, several compilers, (sometimes) linkage editor, file management system, debugging aids and utilities. Such development systems are offered by several semiconductor manufacturers and equipment or specialized manufacturers like Tektronix, Cromemco or IMSAI. Some features and consequences of development systems will be discussed in the following chapters.

7.1 System design

As a difference to the application of general purpose computers, system design of dedicated systems does not start with the selection of the available computers but with the design of the system architecture - possibly at the chip level (see chapter 5). Typical questions are:

- Which processor, which interfaces or which boards should be used?
- Which problems should be handled by hardware, which by firmware and which by software?
- How many subsystems (processors) should be used?
- Which subtasks should be handled by which subsystem?
- Which communication methods (hardware and software) should be used?
- Are subsystems available which can be reused (hardware and software)?

etc.

These questions demonstrate that system design cannot be split into hardware and software design. For this reason two other problems come into view. First of all many multiprocessor systems with very different system architectures have already been designed and successfully implemented for various applications. Nevertheless no general methods and tools for such system design are available up to now. A second point which must be taken into account is that hardware and software specialists worked separately in the past. Even discussions - which happened really seldom - were somewhat awkward due to the different terminology and working methods. This situation makes also the development of design methods and tools harder.

Last but not least some requirements which influence the system architecture are strengthened for dedicated systems:

- the safety and security of the system (hardware and software) including the fulfillment of the required reaction time (real-time behaviour),
- application-oriented test tools for hardware and software development which support the analysis of errors,
- far-reaching automation of application-oriented testsystems (equipment check-out systems) for the system-test at the end of the hardware production as well as for the maintenance,

etc.

Finally, an advantage of dedicated systems should be mentioned. Contrary to general purpose systems the type and the extent of a specific application is well-known. Therefore it is easier in most cases to fulfill the performance requirements by using dedicated systems.

7.2 Coding

The coding of microprocessors itself is very similar than compared to the coding of general purpose computers. However, it must be taken into account that in most cases hardware-dependent programs must be written.

Due to the availability of development systems which are normally used in batch mode, text editors are used instead of forms, a pencil and an eraser. The source programs are stored on the floppy disk which can also be used as input for the assembler or compiler.

Most text editors are teletype-oriented and similar to those well-known of minicomputers. Only the new video-editor of the Mostek development system and very few from equipment manufacturers are CRT-oriented. Instead of commands like I (insert), D (delete), S (substitute), L (lines) etc. the cursor and keys such as roll up, roll down, roll left, roll right are used to change the "text" or to have a look at it.

Macro-assemblers allow the application of macro-techniques, but no macro-libraries which were developed and maintained by manufacturers are available. Intel for example offers INSITE which consists of a large library of very different programs, routines and macros for Intel processors. However, the INSITE library contains only software which was written by users and Intel only organizes this program exchange without any responsibility for the correctness of the software and the corresponding documentation.

7.3 Compiling

The compilation of source programs was already mentioned in chapter 6.2 (assemblers and compilers). Nevertheless some additional remarks are necessary.

Source-programs are compiled on a development system in a hands-on manner. Therefore the compilation time is relevant. Especially for large programs only separate compilations of program modules are economical. During the test or the maintenance only the modules must be recompiled which were changed. In order to be able to program the communication between modules at the source level (data, labels etc.), for instance, the corresponding identifiers must be specified as "external" or "entry". Before loading the different modules (object code) must be linked together to an executable program using a linkage editor. Although linkage editors are very important - especially if high-level languages are used - they are only available in a few development systems.

The error listings and error analysis are often similar to those which are already available in the sixties. For example the type of errors is often given only as a number. The corresponding text must be looked up in a manual. All these weaknesses (missing linkage editors, error handling, library management, macro-libraries etc.) are not a question of methods but of investing manpower for implementation and maintenance. However, this lack of investment by semiconductor manufacturers is understandable because they are

(traditionally) only interested in producing and selling large series of electronic components (diodes, transistors and now IC's). Of course the competition on the market forces them today, also to offer boards, development tools and some system software components.

7.4 Testing

Due to the mutual dependence of hardware and software in the case of dedicated systems, test methods and tools are very important. Often the software must be partially developed in parallel with the hardware.

As a first simple tool design kits (fig. 21) are available to ease the development of prototype boards. They include a small monitor which for instance allows one loading, starting and modifying programs. But such kits (hobbyists are in favour of them) do not allow complete tests from the economical point of view.

The next - much more expensive - step is offered by the already mentioned development systems. These systems are working as host-systems. Direct techniques (the processor of the target and the development system are identical) or cross-techniques are used. Even very capable systems must be expanded for system tests. Also it must be taken into account that the final system test must be executed on the target system which again needs further tools. This situation can be demonstrated by a simple example: a microprocessor-based system which controls a CRT-terminal including two magnetic cassette drives. The software modules support the generation of the screen image (including cursor handling and text-editor functions), keyboard input handling (including function key interpretation), data management functions and teleprocessing drivers. The resident parts of the software will be stored into PROM's. In this case the following difficulties exist during the test:

- no hardware is available for the loading of object programs and display or printout of memory and register contents (because these devices and the corresponding software are under development!),
- PROM's and RAM's for the resident modules are (up until now) not pin-compatible (the final board cannot be used or the program changes during the test are in the case of EPROM's very time-consuming,
- the usage of development systems also requires additional hardware development in order to connect it directly to the screen, the keyboard etc.,
- even simple debugging aids like address-stop are not available.

Most of these difficulties can be avoided by using so-called in-circuit emulators (84) in connection with the development system (fig. 19). In this case the CPU-chip of the target system is replaced by a corresponding connector. The target system is connected with the development system by a cable. A special board in the development system contains an identical CPU-chip which was pulled out of the target system. The development system itself is mainly managing and supporting (debugging aids) the test system. The target system is nearly real-time. In the case of the Mostek development system (85) both CPU's are taken over by the same chip. Therefore the target system runs in real-time. Such in-circuit emulators are offering mainly the following features:

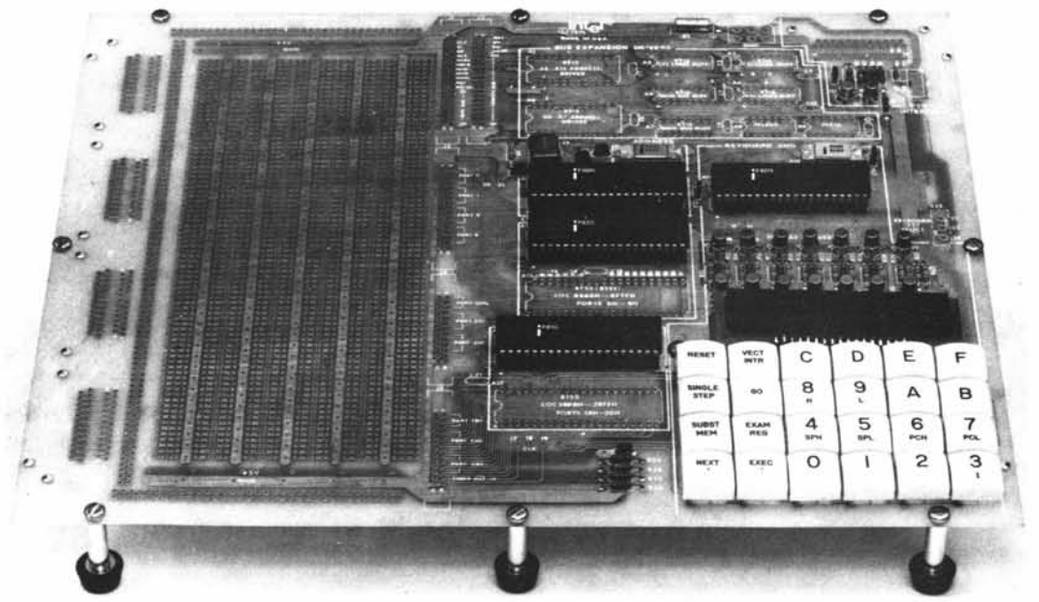

Fig. 21: System design kit (SDK 85)

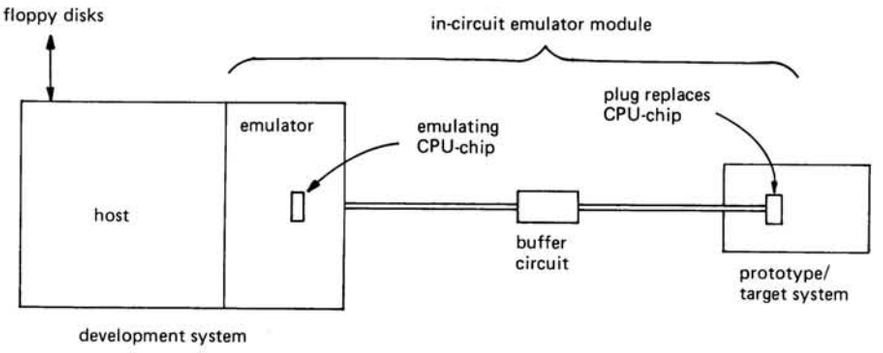

Fig. 19: Development system and in-circuit emulator

- mapping of memory partitions from the target to the development system (replacement of not available ROM's, PROM's etc.),
- tracing of signals on the level of machine-cycles (interfaces, DMA etc.) in order to analyze software errors, and
- general extensive debugging aids for test control.

An in-circuit emulator consists of hardware and software.

In-circuit emulators can also enable a development system to apply cross-techniques during the test. As an extension to a development system which is based on a certain processor (e.g. Mostek on the Z80 (85)), an in-circuit emulator for another processor (in the case of Mostek an F8 in-circuit emulator) is used. Therefore the tests can be executed in a real-time mode without any simulation. However, only in-circuit emulators for their own product line are offered in most cases. For instance it is also possible to test systems based on the Intel 3000 (slice processor) using an Intellec II based on 8080. Only a few manufacturers support others like Mostek (second sourcing!) or Tektronix (no semiconductor manufacturer! (Fig. 20). Tektronix offers in-circuit emulators for 2650, 8080, 8085, Z80, 6800, F8 and 9900 (86).

Development systems are really powerful tools. Nevertheless some problems are left. In the case of a process controller the system test often cannot be performed in a real-time environment. Also the execution of all test cases and the checking of the test results could be too lengthy. For example it is too expensive to test a text-editor completely by using a human operator on the keyboard/display. Therefore environment simulators, test drivers and test-result analyzers are additionally necessary to automate the tests as far as possible. This also enables the programmer for example to reproduce the same system status which was followed by a (hardware or software) error. The same tools can be used during maintenance.

7.5 Software portability

At the beginning of computers only machine-oriented languages were available. Coding and testing was an extremely time-consuming job. All algorithms had to be reprogrammed for each type of computer. An exchange of software between users of different types of computers was not possible.

The main objective of the first generation of general purpose high-order languages (HOL) like FORTRAN, ALGOL 60 and COBOL was the portability of source programs. This means that source programs can be compiled

Fig. 20: Development system (Tektronix 8002 Microprocessor Lab)

for different computers without any changes. More than ten years of experience with this method shows that a "complete portability" can only be reached under certain circumstances. The degree of portability which can be reached for source programs depends mainly on

- the architecture of the computer (especially the I/O organization),
- the operating system which is used as the basis of the run time system,
- the type of application, and
- last but not least the compiler (subset of an HOL, extensions of an HOL, optimizations of object programs etc.).

Additionally different features of peripherals, application-oriented requirements and finally the human behaviour (e.g. to know what things can be done "much better") cause changes of the source programs even if the source program and the compilers offer full portability. This situation stimulated Poole and Waite to define the terms portability (87) and adaptability of programs.

In all discussions about software portability an important assumption is accepted: only general purpose computers are taken into account. Otherwise the necessary investment for compilers, run time systems, HOL-oriented test aids etc. are too high.

Dedicated systems as well as general purpose computers are based on the same electronic components but the architecture of dedicated systems differs from system to system. Also a rapidly increasing number of multiprocessor systems influences the software structure. A general trend of real-time systems in the direction of dedicated systems must be stated. Therefore the advantage of software portability which is left in the area of dedicated systems, consists mainly of the readability of programs.

However, another objective which is related to software portability gains a growing importance. Due to the distribution of subtasks or functions to different (small) processors the chance for a reusage of such system components becomes much higher. These system components like teleprocessing controllers, high-level ADC's, array processors, process controllers, etc. consist of hardware, firmware and software, fixed altogether on a printed circuit board. A very interesting side-effect of this method is that such systems are often reused without any changes. The reason for this is not only the comparitively small size of these systems but much more important is the fact that the software is fixed in read-only memories (static software). They cannot be as easily changed as HOL-programs on source-level! This method is not a new one but has already been successfully used by hardware professionals for many years. However, the consequences of the requirement on the "reusability of system components" must be taken into account already in the system design phase.

8. Economical aspects

It is not possible here to discuss all economical consequences of microprocessors to the architecture and the development of dedicated systems. Therefore only a few aspects have been selected which seem to be important and differ from general purpose computers.

8.1 Manufacturers and distributors

General purpose computers are developed, manufactured, sold and maintained by computer manufacturers. Each specific type is only offered by one company. Exceptions are Itel, Amdahl etc. Only for some components like disk drives, printers, teleprocessing control units, memories etc. the user has the choice to select one of the alternative manufacturers of plug-compatible devices in order to install "mixed hardware".

The situation of microprocessors is another one. Semiconductor manufacturers are the traditional suppliers of computer and peripheral manufacturers. Therefore they usually discuss with potential customers who need thousands of chips a year and take over full responsibility for the design, manufacturing and maintenance of hardware and software. However, this relation is slowly changing. Today semiconductor manufacturers offer already development systems, single-board computers and some system-software components.

A very important difference between computers and electronic components (e.g. microprocessors) is the lifetime of a specific type. The production of a certain type of computer will be stopped a short time after the availability of its successor. This does not hold true for IC's. Successors are often announced a short time after the availability of their predecessors. Nevertheless the "old chips" must be produced for many years because in most cases the redesign of boards, the changeover of the production facilities (hardware and equipment manufacturers!) and the technical service is too expensive. Last but not least spare parts are needed.

Another fact is also very interesting. Similar to other electronic components like tubes, diodes and transistors the customer can buy a chip with the same specifications from different manufacturers: the first source which originally developed an IC and second sources. There are different kinds of second sourcing:

- The second source manufacturer has a full licence contract which possibly

includes the corresponding development system, system software, single-board computers, etc. Also the same technology and masks for the production of chips is used. Such a contract exists for example between Intel and Siemens.

Another type of second sourcing is very interesting which is also based on a licence contract. In this case the second source manufacturer takes over mainly the functional specifications but applies for instance another technology or another combination of functional elements on the chip. For instance Mostek's F8 is functionally a second source for the Fairchild F8 (3850). Additionally Mostek developed its MK3870 as a one-chip version of the F8. A second source for the Mostek MK 3870 is Fairchild! Sometimes manufacturers are also offering their own development systems like Mostek for the Z80 and MK3870.

The present situation of second sourcing is a little bit confusing. Some manufacturers are first source and second source as well. For instance, National Semiconductor offers the SC/MP as first source and many chips from Intel and Signetics as second source. Furthermore some second source manufacturers received licence contracts from strongly competing first source manufacturers (e.g. Mostek for Zilog Z80 and Intel 8086 and AMD for Zilog Z 8000 and Intel 8085). A survey on second sources is given, for instance, in (8).

In addition to semiconductor manufacturers so-called distributors are selling chips. They are able in most cases and willing to offer a better consultation on applications and service for those customers who only need small quantities. Especially for these users it is very interesting that the distributors offer the products of several semiconductor manufacturers. In connection with second sources it is therefore sometimes possible to discuss the advantages and disadvantages of some competing products with only one distributor.

Finally a wide range of manufacturers have been rapidly growing during the last years which use the electronic components or boards in order to design and implement dedicated systems (hardware and software). They consult and support other manufacturers for example in the replacement of discrete logic by microprocessors or work directly for end-users. These so-called system-houses were either founded during the last years or were former software-houses or consultance companies. They take over the design implementation and maintenance of hardware as well as software.

8.2 Some remarks on prices

The low prices of chips (e.g. CPU chip less than 10 US $) or of single-board computers (often less than 500 US $) are sometimes misleading. Only to enable the professionals to program and to test a dedicated system an investment of at least 12.000 US $ for a development system (see chapter 6 and 7) is necessary. Obviously this investment can be used later on in other projects with corresponding processors.

Another point is the relation between the cost for hardware and software. The general trend seems to be that the main part (88, 89) must be paid for the software (80 % and more). This holds only true if a single system for instance based on "standard boards" must be developed. On the other hand in the case of a small or large series of identical systems this situation changes drastically (table 5). This demonstrates again the importance of the requirement on the "reusability of system components".

9. Conclusion

Microprocessors and all the other new electronic components are extremely powerful. An end to the present fast innovative technology phase is not (yet) to

no. of identical systems	hardware (DM/board)	software development (DM)	system HW + SW (DM/system)	software %
1	1 200,–	30.000,–	31 200,–	96.2
10	1 200,–	30 000,–	4 200,–	71.4
50	1 200,–	30 000,–	2 700,–	55.6
100	1 200,–	30 000,–	1 500,–	20.0
1.000	700,–	30 000,–	730,–	4.1
5.000	700,–	30 000,–	706,–	0.9
10.000	700,–	30 000,–	703,–	0.4
hardware: Intel 8085, 512 byte RAM, 4 Kbyte EPROM, 1 serial interface, 38 I/O lines				

Table 5: Relation between hardware and software costs

be seen. Together with the well-known tendency of their prices new application areas will be opened.

On the other hand there are also some negative aspects. The high speed of innovation causes a rapidly growing variety of electronic components and computer elements. The tendency to develop a fast (32-bit) one-chip computer which has 1 Mbyte of memory (or even more) directly available, does not seem to be the solution of the present software crisis (compilers, operating systems, test aids etc.!). It could also inaugurate a new (micro-) version of the same software crisis. Apparently it makes no sense only to implement the well-known and established computer architectures with other elements (under the objective to cheapen hardware prices).

For future research, development and last but not least education, some points should be taken into account. A simple, well-defined partitioning of computer systems in hardware and software with the machine instructions as an "interface" is no longer realistic. In order to develop new system architectures, the software specialists are forced to be more familiar with the basics of hardware and its trends and vice versa. Also a close cooperation between hardware specialists, software specialists and users is absolutely necessary.

Finally in some areas methods and tools are missing.

- Which functions should be implemented on which level (hardware, firmware, software)?
- Which are the criteria for the distribution of functions and subtasks among system components?
- Design methods and tools for microprocessor-based distributed systems are missing.
- Methods and tools which enable the (not computer experienced) user to specify, to describe, to test and to maintain his application in a real application-oriented manner.

This list is obviously not complete. Nevertheless it should motivate computer scientists to critically examine present methods and tools.

REFERENCES

(1) Curron, L. (1977). Personal computers mean business. Electronics March 31, pp. 89-96.

(2) EDP Analyzer (1978). 'Personal' computers for business. EDP Analyzer Vol. 16 No. 5, pp. 1-13.

(3) Adams, W.Th., Smith, S.M. (1978). How bit-sclice families compare. Electronics August 3, pp. 91-98; Electronics August 14, pp. 96-102.

(4) Wilkes, M.V. (1951). The best way to design an automatic calculating machine. Proceedings Manchester University Computer Inaugural Conference, pp. 16-21.

(5) Salisbury, A.B. (1976). Microprogrammable computer architectures. Computer Design and Architecture Series No. 1, Elsevier Computer Science Library, New York.

(6) Bushell, R.G. (1978). Higher-level languages for microprogramming. Euromicro Journal, No. 4, pp. 67-75.

(7) Chu, Y. (ed.) (1975). High-level language computer architecture. Academic Press, New York.

(8) Electronics (1978). Microprocessor data manual. Electronics No. 21, October 11, pp. 53-217.

(9) Electronics (1978). Technology update. Electronics No. 22, pp. 91-217.

(10) Grossman, M. (1978). Second generation 16-bit microcomputers are here - but so are plenty of software problems. Electronic Design No. 23, pp. 60-70.

(11) Katz, B.J., Morse, S.P. et al. (1978). 8086 microcomputer bridges the gap between 8- and 16-bit designs. Electronics, February 16, pp. 99-104.

(12) Shima, M. (1978). Two versions of 16-bit chip span microprocessor, minicomputer needs. Electronics, December 21, pp. 81-88.

(13) Intersil (1977). IM 6100. Intersil data sheet.

(14) Gregory, V., Dellande, B. (1977). Industrial control unit MC 14500 B, Handbook. Motorola 165-177/10K.

(15) Shanks, R.R. (1974). Amorphous semiconductors for electrically alterable memory applications. Computer Design, No. 5, pp. 95-100.

(16) Altman, L. (1977). Memories. Electronics January 20, pp. 81-96.

(17) Hnatek, E.R. (1978). Current semiconductor memories. Computer Design No. 4, pp. 115-126.

(18) Juliussen, J.E., Lee, D.M., Cox, G.M. (1977). Bubbles appearing first as microprocessor mass storage. Electronics August 4, pp. 81-86.

(19) Altman, L. (1978). New arrivals in the bulk storage inventory. Electronics

April 13, pp. 106-113.

(20) Threewitt, B. (1978). CCDs bring solid-state benefits to bulk storage for computers. Electronics June 22, pp. 133-137.

(21) Burton, D.P. (1978). Know a microcomputer's bus structure. Electronic Design, No. 13, pp. 78-84.

(22) Mahr, W., Patzelt, R. (1978). Improvements of multiprocessing capabilities of microprocessor buses. Euromicro Journal, No. 4, pp. 207-219.

(23) Walze, H. (1978). Bit serial communication in decentralized control systems - approaches for common solutions. IFAC Congress Helsinki, Vol. I, pp. 707-710.

(24) Lipovac, V., Oh, S.J. (1977). Design an IEEE-488 bus into an FPLA. Electronic Design, No. 24, pp. 104-111.

(25) Hootman, J. (1978). See how IEEE-488 bus works by designing a compatible A/D - D/A system. Electronic Design, No. 20, pp. 88-91.

(26) Walze, H. (1977). PDV-Bus löst Kopplungsprobleme bei der industriellen Prozeßautomatisierung. Elektronik No. 1, pp. 77-80.

(27) Buxmeyer, E., Haussmann, G., Mielentz, P., Walze, H. (1976). Serielles Bussystem für industrielle Anwendungen unter Echtzeitbedingungen (PDV-Bus). KFK-PDV Bericht Nr. 91, Karlsruhe.

(28) Purdue Europe TC5 (1977). Serial line sharing system for industrial real-time applications (SIR). Purdue Europe TC5, working paper.

(29) Rolander, Th. (1977). Intel Multibus Interfacing. Intel Application Note AP-28.

(30) Mostek (1978). STD-Z80 bus description and electrical specifications. Mostek micro design series, data book no. 79634, pp. 37-40.

(31) Pol, B. (1978). Der S-100 Bus - ein de-facto Standard auf dem Mikrocomputer-markt. Elektronik Vol. 2, pp. 47-51.

(32) Nissim, J. (1978). DMA controller capitalizes on clock cycles to by pass CPU. Computer Design, No. 1, pp. 117-124.

(33) Prazak, P., Mrozawski, A. (1977). Analog I/O hybrids simplify microprocessor interfaces. Electronics May 26, pp. 106-110.

(34) Brokaw, P. (1978). I^2L puts it all together for 10-bit A/D converter chip. Electronics April 13, pp. 99-105.

(35) Waser, S. (1978). State-of-the-Art in high-speed arithmetic integrated circuits. Computer Design, No. 7, pp. 67-75.

(36) Intel (1977) CRT terminal design. Intel Application Note No. AP 32.

(37) Motorola (1977). CRT controller MC 6845. Motorola data sheet.

(38) National (1978). DP 8350 series programmable CRT controller. National semiconductor data sheet.

(39) National (1977). INS1771-1 Floppy disk formatter/controller. National semiconductor data sheet.

(40) Intel (1978). Using the 8273 SDLC/HDLC protocol controller. Intel Application Note No. AP 36.

(41) Weissberger, A.J. (1978). Data-link control chips bringing order to data protocols. Electronics June 8, pp. 104-112.

(42) Meronek, B. (1979). Data-link control chip supports all three bit-oriented protocols. Electronics, No. 2, January 16, pp. 137-142.

(43) Travis, S. (1978). Communication chip interfaces with most microprocessors. Electronics March 6, pp. 123-128.

(44) Trottier, L., Matic, B. (1978). One chip controls keyboard and display. Electronics May 11, pp. 125-130.

(45) Marathe, A.D., Chandra, A.K. (1978). Hardware/software for process control I/O. Computer Design, No. 3, pp. 122-126.

(46) National semiconductor (1975). Simple cost-effective microprocessor SC/MP Data Sheet.

(47) Peuter, B.L., Prosenko, G.J. (1978). One-chip microcomputer excels in I/O- and memory-intensive uses. Electronics August 31, pp. 128-136.

(48) Dozier, W., Green, R.S. (1978). Single-chip microcomputer expands its memory. Electronics May 11, pp. 105-110.

(49) Benson, T. (1978). Microcomputers: single-chip or single-board? Electronic Design, No. 13, pp. 86-91.

(50) Deverse, Frank (1977). Consider Mask-programmable arrays instead of custom-designed circuits. Electronic Design, No. 26, pp. 156-159.

(51) Reyling, G. (1975). PLA's enhance digital processor speed and cut component count. Electronics Book Series Microprocessors, pp. 87-92.

(52) Beaston, J. (1978). Second-generation microcontrollers take on dedicated-function tasks. Electronics November 23, pp. 127-132.

(53) Intel (1978) SBC 80/30 single board computer hardware reference manual. Intel 980061.

(54) Mostek (1977) SDB-80E Software development board. Mostek MK78548.

(55) Siemens (1978). Kurzbeschreibung SMP809. Siemens B1970.

(56) Dolch, V. (1978). Logikanalyse-Gerätetechnik und Anwendungen. Elektronik Vol. 3, pp. 40-46.

(57) Bodley, N., Burski, D., Schindler, M. (1978). Microcomputer data manual. Electronic Design, No. 11, pp. 65-226.

(58) Nicoud, J.D. (1976). A common microprocessor assembly language. Euromicro Symposium preprints, North-Holland, pp. 213-219.

(59) Brinkmann, K.-D., Kreissel, F. (1978). Ein universell einsetzbarer Assembler. Elektronik 11, pp. 81-84.

(60) Antoy, S., Cordano, F., et al. (1978). GPCA: a general purpose cross-assembler. Euromicro Journal, No. 4, pp. 222-226.

(61) Schmit, A.M., Schmit, P., Berthoud,M. (1977). A universal cross-assembler for microprocessors. Euromicro Symposium preprints, North-Holland, pp. 112-122.

(62) Musstopf, G. (1976). Implementation languages - definition and area of use. Purdue Europe TC-4 working paper.

(63) Intel (1977). PL/M-80 Programming manual. Intel 98-268B.

(64) McCracken, D. (1978). A guide to PL/M programming for microcomputers applications. Addison-Wesley.

(65) Intel (1976). ISIS-II system user's guide. Intel 98-306.

(66) Corti, P., Daprà, A., Crespi-Reghizzi, S. (1978). High-level microprocessor oriented languages. Euromicro 78 Symposium, North-Holland, pp. 118-127.

(67) Posa, J.G. (1979). Programming microcomputer systems with high-level languages. Electronics, No. 2, January 18, pp. 105-112.

(68) Jensen, K., Wirth, N. (1974). PASCAL-user manual and report. Lecture notes in computer science 18, Springer Verlag.

(69) Lewis, A. (1978). Industrial real-time BASIC. Purdue Europe TC2 working paper, IRTB-E 78/9.

(70) ISA-S61.1 (1975). Standard Industrial Computer Systems. FORTRAN procedures for executive functions, process I/O, and bit manipulation.

(71) Hansen, P.B. (1975). The programming language concurrent PASCAL. IEEE Trans. Software Engineering, Vol. 1, No. 2, pp. 199-207.

(72) Musstopf, G., Orlowski, H., Tamm, B. (1979). Program generators for process control applications. IFAC/IFIP SOCOCO 1979, Prague.

(73) Field, W.B., Rehnvorg, F.M. (1975). Implementation of problem-oriented languages using a microprocessor. Minutes of the 1975 Regional Meetings International Purdue Workshop, pp. 127-197.

(74) Tektronix (1978). 8002 uP Lab system user's manual. Tektronix 070-2313.

(75) Mostek (1978). SFP-80 DOS, Operating Manual. Mostek MK 78557.

(76) Burzio, G. (1978). Operating systems enhance microcomputers. Electronic Design, No. 13, pp. 94-99.

(77) Thorelli, L.-E. (1978). Design and Implementation of a small real-time monitor. IFAC/IFIP Workshop on Real-Time Programming, Mariehamn.

(78) Kratzer, G., Schrott, G. (1978) Interfacing real-time operating systems to process control languages. IFAC/IFIP Workshop on Real-Time Programming, Mariehamn.

(79) Wirth, N. (1977). Modula: a language for modular multiprogramming. Software-Practice and Experience Vol. 7, pp. 3-84.

(80) Burgett, K., O'Neil, E.F. (1977). An integral real-time executive for microcomputers. Computer Design, No. 7, pp. 77-82.

(81) Intel (1978). RMX/80 User's guide. Intel 9800522B.

(82) Baumann, R. (1978). Computer aided design and implementation of control algorithms. IFAC Congress, Helsinki, Vol. I, pp. 649-655.

(83) Winkler, J.F.H., Stoffel, C. (1978).

Methode zur Betriebssystemgenerierung und -Modularisierung für Prozeß-rechensysteme. KfK-PDV 153, Karlsruhe.

(84) Yen, M.Y. (1977). Fast emulator debugs 8085-based microcomputers in real-time. Electronics, July 21, pp. 108-111.

(85) Mostek (1978) AIM-80 A, application interface module, operators manual. Mostek MK 7859.

(86) Tektronix (1978). 8002 uP Lab-Z80 assembler and emulator user's manual. Tektronix 070-2415.

(87) Poole, P.C., Waite, W.M. (1973). Portability and adaptability. Lecture Notes in Economics and Mathematical Systems Springer Verlag.

(88) Phister, M.Jr. (1978). Analyzing computer technology costs. Computer Design, No. 7, pp. 91-98.

(89) Schindler, M.J. (1978). Fill your microcomputer with the right software package. Electronic Design, No. 14, pp. 64-72.

vendor	part no.	word size (bit)	RAM	ROM/ PROM	no. of basic instructions	instruction time (us)	no. of I/O lines	no. of pins
AMI	S 2000	4	64 x 4	1024 x 8	51	4.5/9	79	40
Intel	804	8	64 x 8	1024 x 8	96	2.5/5	27	40
Mostek	3872	8	64 x 8	2048 x 8	70	1/6.5	32	40
Rockwell	R 6500/1	8	64 x 8	2048 x 8	56	1/3.5	32	40
Zilog	Z8	8	124 x 8	2048 x 8	47	1.5/3.75	32	40
TI	9940	16	128 x 8	2048 x 8	68	2.4/5	16	40

Table 3: Single chip systems

vendor	part no.	word size (bit)	no. of basic instruction	instruction time (us)	no. of registers	no. of pins
Fairchild	F8	8	69	2/13	64	40
Intel	8085	8	80	0.8/5.2	8	40
Motorola	6803	8	82	2/12	128	40
NS	SC/MP II	8	46	5/10	0	40
RCA	1803	8	91	2.5/3.75	16	28
Signetics	2650	8	75	1.5/6	7	40
Zilog	Z80	8	150	1/5.75	14	40
Ferranti	F 100 L	16	153	1.19/14	0	40
Intel	8086	16	97	0.4/37.8	8	40
Motorola	68000	16	61	0.5/?	16	64
TI	SBP 9900	16	69	2/31	16	64
Zilog	Z 8000	16	110	0.75/90	16	40

Table 4: General purpose microprocessors